VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Informační systém pro analýzu dat poruch v elektrických sítích

Information System for Analysis of Failures in Power Networks

2019 Tomáš Gluszny

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Gluszny**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Informační systém pro analýzu dat poruch v elektrických sítích
Information System for Analysis of Failures in Power Networks

Jazyk vypracování: čeština

Zásady pro vypracování:

Na Katedře informatiky je vyvíjen informační systém pro analýzu dat poruch v elektrických sítích, který je využíván distributory elektrické energie v České i Slovenské republice. Cílem práce je reimplementace datové vrstvy a formulářů tohoto informačního systému.

1. Nastudujte stávající informační systém.
2. Přepište datovou vrstvu pro zvolený databázový systém podporující SQL.
3. Přepište uživatelské rozhraní tak aby odpovídalo moderním trendům.
4. Výsledné řešení otestujte a proveďte zhodnocení úprav.

Seznam doporučené odborné literatury:

[1] GOŇO, Radomír, Michal KRÁTKÝ a Stanislav RUSEK. Analysis of Distribution Network Failure Databases. Przeglad elektrotechniczny. Warszawa: SIGMA-NOT, 2010, 86(8), s. 168-171. ISSN 0033-2097.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Michal Krátký, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 26. dubna 2019 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 26. dubna 2019 .....

Rád bych poděkoval doc. Ing. Michalovi Krátkému, Ph.D. za vstřícnost, podnětné připomínky a odborné vedení během mé práce.

Abstrakt

Na Katedře informatiky je vyvíjen informační systém pro analýzu dat poruch v elektrických sítích, který je využíván distributory elektrické energie v České i Slovenské republice. Cílem práce je reimplementace datové vrstvy, uživatelského rozhraní a nasazení informačního systému na dva rozdílné databázové systémy SQL Server a SQLite, jelikož informační systém doposud využíval databázový systém Radegast DB bez podpory SQL.

Klíčová slova: informační systém, databáze poruch, analýza, elektrické sítě, výpočet spolehlivosti, SQL

Abstract

An information system for analysis of failures in power networks is being developed at the Department of Computer Science. The information system is used by electricity distributors in the Czech Republic and Slovak Republic. The aim of the thesis is to re-implement the data layer, user interface and deploy the information system for two different database management systems SQL Server and SQLite, because the information system currently uses the Radegast DB without SQL support.

Key Words: information system, database of failures, analysis, power networks, reliability computation, SQL

Obsah

Seznam použitých zkratek a symbolů

IS	–	Information System
DBMS	–	Database Management System
SQL	–	Structured Query Language
T-SQL	–	Transact-SQL
PL/SQL	–	Procedural Language/SQL
PL/pgSQL	–	Procedural Language/PostgreSQL
ANSI	–	American National Standards Institute
ISO	–	International Organization for Standardization
API	–	Application Programming Interface
CRUD	–	Create Read Update Delete
CSV	–	Comma Separated Values
XML	–	Extensible Markup Language
JSON	–	JavaScript Object Notation
BSON	–	Binary JSON
MS-DOS	–	Microsoft Disk Operating System
EULA	–	End User License Agreement
BSD	–	Berkeley Software Distributio
GNU GPL	–	GNU General Public License
GNU AGPL	–	GNU Affero General Public License
ORM	–	Object Relation Mapping
DAO	–	Data Access Object
HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je se seznámit se stávající verzí informačního systému pro analýzu dat poruch v elektrických sítích, včetně implementace, kterou je potřeba nahradit současnými technologiemi, technikami a funkcemi. Hlavním úkolem je přepsat datovou vrstvu pro relační databázové systémy SQL Server a SQLite a následně porovnat rozdíly. Výsledné řešení je nutno otestovat a provést zhodnocení úprav systému. Finální verze aplikace bude nasazená na školní webový server.

Důvodem přepracování stávajícího informačního systému je snaha nahradit současný embedded databázový systém Radegast DB na relační databázový systém (anglicky zkráceně DBMS) podporující jazyk SQL. Následně je potřeba upravit uživatelské rozhraní, tak aby odpovídalo moderním trendům.

Bakalářská práce je rozdělená do 5 kapitol. V kapitole 2 je obsah zaměřen na úvod do databázových systémů, datové modely a na další nezbytné teoretické pojmy související s informačním systémem. Kapitola 3 je věnována samotnému informačnímu systému. Popisuje jak jsou data ukládána a jak se následně zpracovávají při výpočtech spolehlivostních parametrů. Kapitola 4 slouží jako dokumentace zachycují postup při reimplementaci systému včetně funkční a datové analýzy. Závěr kapitoly je zaměřen na testování, zhodnocení úprav systému a srovnání výsledků, především z hlediska přístupové doby použitých databázových systémů.

2 Úvod do databázových systémů

2.1 Databáze

Databáze slouží jako úložiště vzájemně propojených dat v počítačově zpracovatelné formě. Ve spojení s informačními technologiemi se databázový systém stává nezbytným nástrojem pro vývoj a údržbu informačních systémů, které zpracovávají data (sdílení, ukládání, vyhledávání, manipulování, zobrazování). Způsob uchovávání informací se vyvinul od síťových a hierarchických datových modelů až po relační modely, ve kterých byla odstraněna řada nedostatků a staly se tak robustní a perspektivní na další desítky let dopředu. Načítat data z databáze lze pomocí příkazů příslušného dotazovacího jazyka, jehož výsledkem je podmnožina z uložených záznamů. Výsledné údaje lze třídit, popřípadě seskupit. Aby bylo možné data co nejefektivněji ukládat, prohledávat nebo zpracovávat, musí být vhodně uspořádaná a organizovaná, čehož lze docílit dodržováním normálních forem [1][2].

2.2 Datové modely

Plochá databáze neboli prostý databázový soubor je považován za nejprimitivnější typ databáze. Data v ní jsou uchovávaná jako sada záznamů. Prostý databázový soubor může být textový nebo binární, ve kterém nejsou žádné strukturální vazby mezi záznamy [1].

Následující datové modely jsou hierarchické a síťové. Hierarchický databázový systém je postavený na stromovém modelu. Každý záznam představuje uzel ve stromové struktuře, kde vzájemný vztah mezi záznamy je typu rodič-potomek. Tato struktura umožňuje přiřadit jednotlivým tabulkám jinou rodičovskou tabulku. Obecně platí, že každý záznam v tabulce musí obsahovat odkaz na jeden ze záznamů z hierarchicky vyšší tabulky. Hierarchický model je tedy schopen modelovat pouze vztahy s kardinalitou 1:N. Vztahy kardinality M:N je možné modelovat za využití dvou vztahů 1:N. V případě síťového databázového systému může být záznam spojený s libovolným počtem dalších záznamů [1].

Relační datový model je v současnosti nejpoužívanější datový model. Pojem relační model definoval v roce 1970 doktor Edgar Codd v článku *A relational data model for large shared data banks* [6]. Relační databáze může obsahovat libovolný počet tabulek, ve kterých jsou záznamy uloženy na řádcích [13]. Každé relační schéma má své atributy, kterým může programátor přiřadit datový typ, integritní omezení nebo jiné podmíněné formátování. Příkladem je primární klíč, který určuje unikátní hodnotu ve sloupci v rámci všech řádků tabulky. Tabulky mohou obsahovat i cizí klíče. Pomocí cizího klíče dokážeme definovat, že se řádek tabulky odkazuje na jiný řádek jiné tabulky. Relační model přináší celou řadu výhod, zejména možnost snadného definování a zpracování vazeb. S relačními databázovými systémy je úzce spojen dotazovací jazyk SQL, neboli strukturovaný dotazovací jazyk. Jeho základní model je obecně použitelný pro naprostou většinu relačních databázových systémů. Od svého vzniku prošel několika revizemi

a poskytovatelé databázových systémů jej obohatili o různá lokální rozšíření, která nemusí být vzájemně kompatibilní [1].

Databázové systémy založené na objektovém modelu ukládají data ve formě objektů. Stejně jako v objektově orientovaných programovacích jazycích, lze uplatnit zapouzdření a dědičnost. Objekty mají vlastní atributy a místo řádků se ukládají samotné instance objektů. Objektové databázové systémy se snaží nabídnout snadnější přístup k datům, jelikož lze k objektům přistupovat přímo v programovacím jazyku skrze metody. Nicméně v současné době existují profesionální nástroje, které dokážou relativně snadno reprezentovat data z relačního databázového systému na objekty [1].

NoSQL je databázový koncept, který využívá jiné techniky než SQL [8]. Nejúspěšnější NoSQL systémy jsou založeny na principu klíč-hodnota, čímž je zajištěna vysoká rychlost zápisu a čtení. Na rozdíl od relačních DBMS není možné vyhledávat atributy podle jejich hodnoty, jelikož se k datům přistupuje vždy prostřednictvím klíče. Hlavní nevýhodou je udržování integrity a absence transakcí. Dalším typem NoSQL jsou dokumentové databázové systémy, které ukládají hodnoty v hierarchické struktuře jako je například XML nebo JSON, přičemž na rozdíl od databázových systémů klíč-hodnota umožňují dotazování nad atributy v rámci hierarchické struktury. Pro dotazování nad XML daty se používá nejčastěji dotazovací jazyk XQuery. Hlavním představitelem dokumentové databáze je databázový systém Mongo DB [24], který ukládá data ve formátu BSON. Mongo DB implementuje vlastní dotazovací jazyk mongo shell umožňující práci s těmito dokumenty. Existují také sloupcově orientované NoSQL databázové systémy, které používají stejně jako relační databázové systémy sloupce a řádky, avšak jejich význam není zcela totožný. Základním prvkem pro ukládání dat je sloupec. Sloupec se skládá z názvu sloupce a hodnoty. Řádek se pak skládá z množiny určitých sloupců. Ve srovnání s řádkově orientovaným relačním datovým modelem dokáže sloupcově orientovaný DBMS zvýšit výkon čtení tím, že nedochází ke čtení všech řádků, avšak výkonnost dotazu je zvýšena pouze při určitém pracovním vytížení. Sloupcově orientované systémy používají stejně jako relační DBMS pro manipulaci s daty dotazovací jazyk, který na rozdíl od SQL nepodporuje operaci *JOIN*. Příkladem sloupcově orientovaného DBMS je Apache Cassandra, využívající jazyk Cassandra Query Language [14].

2.3 Architektura databázových systémů

2.3.1 Klient-Server

Klient-server je softwarová architektura, oddělující klienta a server, kteří spolu komunikují přes počítačovou síť. Tento typ architektury se používá ve většině databázových systémů, protože umožňuje přístup k datům na serveru více uživatelům současně. Zároveň je vyřešena i ochrana dat před zneužitím, jelikož přístup je možný pouze prostřednictvím programu databázového systému. Vzhledem k tomu, že se data ukládají centralizovaně na serveru, odpadá potřeba aktualizovat data na každé stanici zvlášť. Možnou nevýhodou je výpadek serveru, přičemž žádosti uživatelů nemohou být splněny. Dalším problémem je přetěžování sítě. Pokud počet požadavků klientů na daný server překročí únosnou mez, může se server snadno přetížit [4].

2.3.2 Embedded

Embedded systémy jsou pravým opakem architektury klient-server. Místo aby se přistupovalo k datům na databázovém serveru, nacházejí se databáze v adresním prostoru aplikace. Dříve se tyto databázové systémy používaly výhradně, jelikož operační systém MS-DOS byl jednouživatelský a počítačové sítě zpočátku téměř neexistovaly. Embedded DBMS je v podstatě knihovna, kterou lze přilinkovat k aplikaci. Aplikace přistupuje k datům prostřednictvím knihovny přímo k souborům s daty. Tento typ se používá obvykle pro práci v single-user režimu.

Příkladem embedded je databázový systém SQLite, který ani zdaleka neposkytuje všechny možnosti, kterými disponuje většina současných databázových systémů. SQLite poskytuje pouze nezbytné základy z jazyka SQL, jako jsou například příkazy pro manipulaci s daty nebo příkazy pro dotazování. Mezi výhody patří odladěnost a nulová údržba, protože se nevyžaduje údržba na straně serveru.

Nevýhodou embedded systémů je absence přístupových práv založených na autorizaci uživatele. Vzhledem k tomu, že uživatel má přímý přístup k datům, postrádalo by to smysl. Pokud je potřeba řešit přístupová práva k datům, řeší se na aplikační úrovni. Existují i jiná rizika. Jelikož embedded databáze sdílejí s aplikací stejný adresní prostor, může aplikace omylem přepsat část paměti, což obvykle ke ztrátě dat [5].

2.4 SQL

SQL je strukturovaný dotazovací jazyk, který je podporován naprostou většinou relačních databázových systémů. První verze tohoto jazyka, nesoucí jméno Sequel, vznikla v roce 1974 ve společnosti IBM. Cílem bylo poskytnout standardní metodu přístupu k datům uloženým v databázovém systému. V první polovině 80. let se na trhu objevily čtyři desítky komerčních SQL systémů. Organizace ANSI zavedla roku 1986 jako standard variantu společnosti IBM. Tento standard bývá označován jako SQL86 [28]. O rok později byl přijat také v ISO. Roku 1989 pak následoval další standard, který umožnil definovat integritní omezení. V roce 1992 dochází k zavedení standardu SQL92 [29]. Nicméně i přes veškeré zavedené standardy si poskytovatelé databázových systémů implementují své vlastní konstrukce, tudíž přenositelnost mezi databázemi je v některých případech omezená [9].

SQL podporuje přidávání, mazání a aktualizování záznamů, včetně dotazování nad záznamy. Jazyk SQL lze rozdělit na podskupiny podle typu příkazů. Mezi příkazy pro manipulaci s daty patří *INSERT, UPDATE, DELETE*. Příkazy pro definici dat slouží k modifikaci objektů jako jsou tabulky nebo uložené procedury a funkce. Zástupci těchto příkazů jsou *CREATE, ALTER, DROP*. Pokud se chce uživatel nad záznamy dotazovat, využije příkaz *SELECT*. Konstrukce *WHERE* slouží pro filtrování záznamů. Vybrané záznamy může seřadit nebo seskupit pomocí konstrukcí *ORDER BY* a *GROUP BY*. Příkazy pro správu databázových transakcí jsou následující: *BEGIN TRANSACTION, COMMIT, ROLLBACK*. Příkazem *GRANT* lze přidělit oprávnění k objektům uživateli a naopak příkazem *REVOKE* dojde k odnětí práv uživatele [15].

2.5 Systémy řízení báze dat

Databázový systém tvoří mezivrstvu mezi aplikacemi a databází. Úkolem DBMS je poskytnout efektivní možnosti pro ukládání a správu dat. Současné databázové systémy řeší autentizaci uživatelů a jejich práva k operacím nad objekty, integritu dat, transakce nebo profilování. Následující text je zaměřen na nejpoužívanější systémy řízení báze dat [1].

MS SQL Server [11] je relační DBMS podporující objektové relační datové modely, který vyvíjen společností Microsoft a je postavený na architektuře klient-server. Systém je založený na jazyce SQL a podporuje procedurální jazyk T-SQL. Mezi další výhody patří škálovatelnost, bezpečnost, robustnost, podpora transakcí, XML a JSON. Microsoft nabízí integrované prostředí SQL Server Management Studio, které disponuje řadou funkcí a nástrojů, které lze použít při vývoji a správě databází. Umožňuje například monitorování a diagnostikování pomalu běžících dotazů a jejich krokování k nalezení příčiny problému. Současné verze jsou dostupné i pro platformu Linux. Poslední vydanou verzí je Microsoft SQL Server 2017 [10].

Oracle Database [16] se řadí mezi nejpoužívanější DBMS na trhu. Stejně jako SQL Server patří mezi komerční, multiplatformní, relační databázové systémy s podporou procedurální nadstavby. Pomocí rozšíření PL/SQL, lze kromě standardních příkazů SQL, vytvářet proměnné, uložené procedury, funkce nebo kurzory. Oracle nabízí bezplatné integrované vývojové prostředí

SQL Developer, které zjednodušuje vývoj a správu databází. V roce 2018 byla vydána poslední verze Oracle Database 18c [17].

Dalším databázovým systémem je PostgreSQL [18], který je užíván především díky open source licenci. Na jeho vývoji se podílí globální komunita vývojářů a firem. PostgreSQL se řadí mezi objektově-relační DBMS, který je dostupný na většinu platforem. PostgreSQL je primárně vyvíjen pro operační systém Linux, respektive pro unixové systémy obecně, nicméně existují i balíčky pro platformu Windows. PostgreSQL se vyvinul z projektu Ingres na Kalifornské univerzitě v Berkeley. Stejně jako předchozí databázové systémy podporuje procedurální programovací jazyk. V případě PostgreSQL se jedná o PL/pgSQL připomínající PL/SQL z Oracle. PostgreSQL nabízí administrační rozhraní pgAdmin, popřípadě webovou variantu phpPgAdmin, které slouží ke správě databázového serveru. Mezi další výhody patří předdefinované funkce, škálovatelnost nebo podpora JSON formátu [19].

MySQL [20] je relační databázový systém, vytvořený švédskou firmou MySQL AB, který je nyní vlastněný společností Oracle Corporation a spadá pod freeware licenci s možností připlatit za pokročilé funkce. MySQL se využívá zejména ve webových aplikacích. Podobně jako u předchozích databázové systémy je multiplatformní, podporující další rozšíření jazyka SQL [21].

V kapitole o architekturách bylo zmíněno, že se SQLite [22] řadí mezi embedded DBMS. Tento DBMS se užívá především v desktopových aplikacích, které nevyžadují přístup k datům na serveru. Celá databáze je umístěna v jediném souboru na disku. Ve srovnání s jinými databázovými systémy, se jedná o odlehčenou variantu, tudíž se programátor musí připravit na její nedostatky. Nelze provádět příkazy pro definici dat, a proto je nutné změny tabulek obcházet přes nástroje, které dokážou spravovat SQLite databázi. Operace *OUTER JOIN*, sloužící pro spojení tabulek může být pouze ve formě *LEFT*. Další nevýhodou je absence uložených procedur nebo uživatelských oprávnění [23].

Posledním uvedeným databázovým systémem je MongoDB [24], který na rozdíl od všech předchozích databázových systémů nepatří mezi relační DBMS, nýbrž mezi NoSQL DBMS. Jedná se o multiplatformní open source software vytvořený organizací MongoDB Inc. Místo relačních tabulek a SQL příkazů, využívá dokumenty podobné formátu JSON. MongoDB patří mezi nejpopulárnější NoSQL databázové systémy [25].

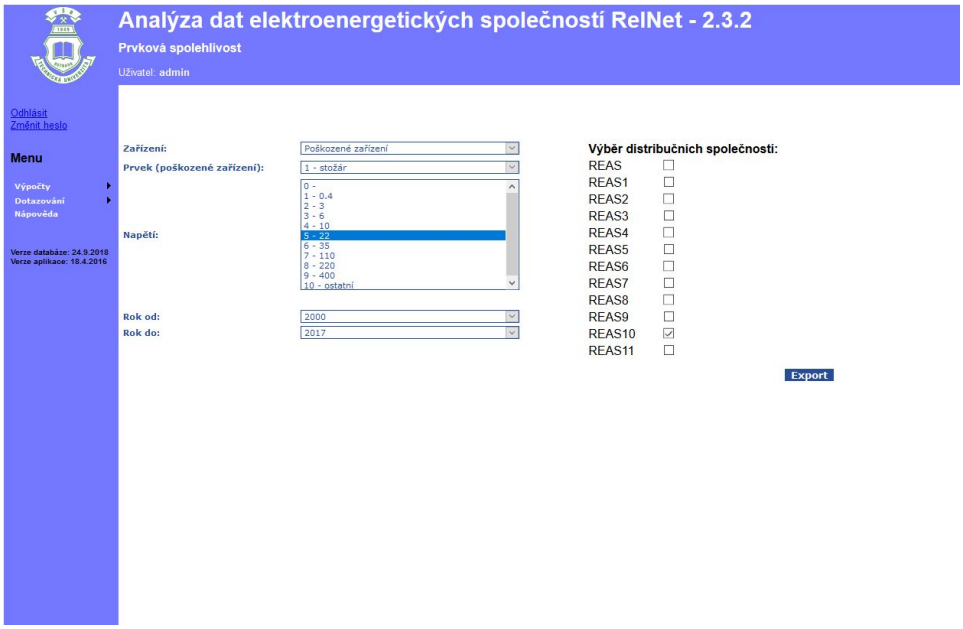
DBMS	Skriptovací jazyk	Typ	Vývojář	Licence
MS SQL Server	T-SQL	objektově-relační	Microsoft	EULA
Oracle DB	PL/SQL	objektově-relační	Oracle Corporation	EULA
PostgreSQL	PL/pgSQL	objektově-relační	PostgreSQL GDG	BSD
MySQL	MySQL	relační	Oracle Corporation	GNU GPLv2
SQLite	-	relační	D. Richard Hipp	volné dílo
MongoDB	JavaScript	NoSQL	MongoDB Inc.	GNU AGPL v3

Tabulka 1: Srovnání 6 vybraných DBMS podle webu db-engines.com [3]

3 Informační systém pro analýzu dat elektroenergetických společností RelNet

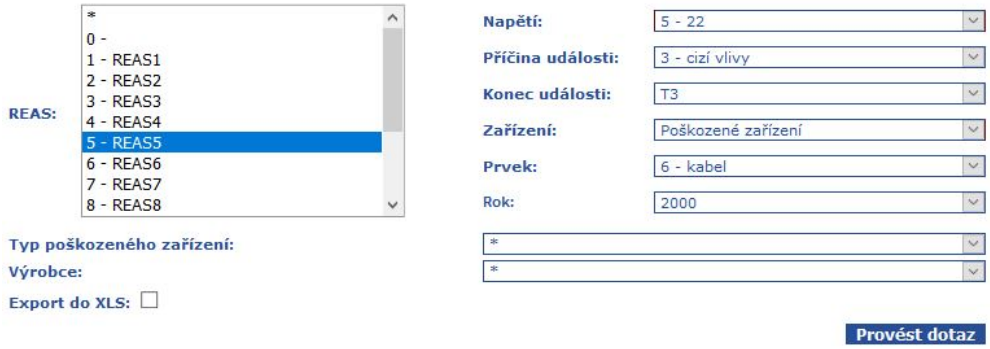
3.1 Základní informace

Na Katedře informatiky je vyvíjen informační systém pro analýzu dat poruch v elektrických sítích, který je dostupný na adrese `https://relnet.vsb.cz/relnet-beta/`. Tento IS umožňuje provádět výpočty spolehlivostních parametrů nad prvky jako jsou pojistky, jističe, transformátory a další. Cílem systému je poskytnou informace o událostech v distribučních sítích dodavatelům elektrické energie. Na základě výsledků se distributoři zaměřují na problematické prvky sítě s cílem zajistit spolehlivé dodávky elektrické energie spotřebitelům. Statistická správnost výpočtů závisí na počtu záznamů v databázi. Větší databáze zachycuje reálný stav prvků v elektrických sítích, proto je nutné sloučit databáze různých distributorů. Hlavním problémem poskytnutých dat je jejich heterogenita: databáze různých distributorů se od sebe navzájem liší, takže bylo vytvořeno univerzální schéma umožňující načítání rozdílných databází různých distributorů. Do databáze můžeme přidávat nové datové sady stávajícího distributora popřípadě nového distributora. Současná databáze obsahuje více než 300 000 záznamů poruch. Ačkoliv se tento informační systém používá pro výpočet spolehlivostních parametrů v České a Slovenské republice, stejně tak by byl použitelný i pro distribuční společnosti z jiných zemí. Výstupem systému je seznam prvků a jejich spolehlivostní parametry, průměrné trvání výpadku zařízení, četnost trvání poruchy nebo nejčastější příčiny poruchy [7]. Na obrázku 1 je uvedená webová aplikace informačního systému. Na obrázku 2 je zobrazeno dotazování nad poruchami.



Obrázek 1: Informační systém pro analýzu dat poruch v elektrických sítích.

Response time: 0,004s



Výsledek:

Trvání výpadků, 0-T0:	94,13h
Střední doba trvání výpadků ze záznamů, 0-T0:	1,96h
Počet záznamů:	48
Počet zobrazených záznamů:	0

1	2	3	4	5	6	7	8	9	10	11	12	13
REAS	Číslo události	Typ události	Rozvodna	Napájecí oblast	Druh sítě	Napětí sítě	Napětí zařízení	Číslo původní události	Příčina události	Druh zařízení	Poškozené zařízení	Typ poškozeného zařízení
5	18044	1	-E-	-E-	-E-	5	5	-E-	33	3	6	-E-
5	16285	1	-E-	-E-	-E-	5	5	-E-	33	3	6	-E-
5	12041	1	-E-	-E-	-E-	5	5	-E-	34	3	6	-E-
5	14840	1	-E-	-E-	-E-	5	5	-E-	34	3	6	-E-
5	16663	1	-E-	-E-	-E-	5	5	-E-	33	3	6	-E-
5	13315	1	-E-	-E-	-E-	5	5	-E-	34	3	6	-E-
5	12795	1	-E-	-E-	-E-	5	5	-E-	34	3	6	-E-
5	14573	1	-E-	-E-	-E-	5	5	-E-	34	3	6	-E-
5	14955	1	-E-	-E-	-E-	5	5	-E-	34	3	6	-E-

Obrázek 2: Dotazování nad poruchami.

3.2 Vybrané atributy pro analýzu dat

Každý distributor poskytuje heterogenní data o poruchách, přičemž je nelze do databáze uložit přímo. Řešením problému je vytvoření společného schématu. Tabulka 2 ukazuje, atributy tohoto schématu. Jakmile jsou data uložena v tabulce, je možné se nad nimi dotazovat [7].

Pořadí	Atribut	Datový typ	Popis	Číselník
1	REAS	NUMBER	Distribuční společnost	c_01
2	Číslo události	CHAR	Identifikační číslo události	
3	Typ události	NUMBER	nahodilá, plánovaná, vynucená	c_02
4	Rozvodna	NUMBER	jednosystémová, dvousystémová...	c_03
5	Napájecí oblast	CHAR	Specifikace oblasti	
6	Druh sítě	NUMBER	izolovaná, kompenzovaná, odporová...	c_05
7	Napětí sítě	NUMBER	3 kV, 110 kV, 220 kV...	c_04
8	Napětí zařízení	NUMBER	3 kV, 110 kV, 220 kV...	c_04
9	Číslo původní události	CHAR	Identifikační číslo původní události	
10	Příčina události	NUMBER	příčiny před započetím provozu, cizí vlivy...	c_06
11	Druh zařízení	NUMBER	venkovní vedení jednoduché, kabelové vedení silové...	c_07
12	Poškozené zařízení	NUMBER	vodič, kabel, transformátor...	c_08
13	Typ poškozeného zařízení	NUMBER	dřevěný sloup, ocelová tyč...	c_10
14	Množství	NUMBER	Množství poškozeného zařízení	
15	Druh zkratu	NUMBER	zkrat jednofázový zemní, zkrat dvoufázový zemní...	c_09
16	Výrobce	NUMBER	Siemens, ABB...	c_11
17	Rok výroby	NUMBER	Rok výroby zařízení	
18	T0	DATE	První výskyt události	
19	T1	DATE	Zahájení opravných prací	
20	T2	DATE	Ukončení opravných prací	
21	T3	DATE	Obnovení dodávky elektrické energie pro všechny spotřebitele	
22	T4	DATE	Ukončení manipulace na zařízení	
23	TZ	DATE	Doba zkratu	
24	P1	NUMBER	Síť bez napětí na začátku události	
25	P2	NUMBER	Síť bez napětí na konci události	
26	D1	NUMBER	Odstavené transformátory na začátku události	
27	D2	NUMBER	Odstavené transformátory na konci události	
28	Z1	NUMBER	Postihnutí odběratelé na začátku události	
29	Z2	NUMBER	Postihnutí odběratelé na konci události	
30	LxT	NUMBER	Počet postihnutých odběratelů vynásobený délkou poruchy	
31	Druh poruchy	NUMBER	bez poškození zařízení, spojené s poškozením zařízení	c_13

Tabulka 2: Vybrané atributy pro analýzu dat [7].

3.3 Číselníky

Některé atributy uvedené ve schématu jsou cizí klíče. Pro takové atributy máme definované číselníky. Atribut pak může nabývat pouze hodnot z příslušného číselníku. Číselníky v České republice jsou vydávány obvykle Energetickým regulačním úřadem [30]. Záznamy v číselníku tvoří pár klíč a hodnota. Kompletní číselníky jsou součástí přílohy bakalářské práce, popřípadě jsou dostupné přímo ve webové aplikaci pod záložkou nápověda [7].

ID číselníku	Název
c_01	REAS
c_02	Typ události
c_03	Rozvodna
c_04	Napětí
c_05	Druh sítě
c_06	Příčina události
c_07	Druh zařízení
c_08	Poškozené zařízení
c_09	Druh zkratu
c_10	Typ poškozeného zařízení
c_11	Výrobce
c_12	Souhrnné údaje
c_13	Druh poruchy

Tabulka 3: Seznam číselníků.

Klíč	Hodnota
1	Izolovaná
2	Kompenzovaná
3	Odporová
4	Kombinovaná
5	Účinně uzemněná

Tabulka 4: Číselník c_05 druh sítě.

3.4 Výpočet spolehlivostních parametrů

Většina distributorů vytváří statistiky pro poruchy, ať už nahodilé nebo způsobené v důsledku údržby. Tyto informace o poruchách jsou následně uloženy do databáze informačního systému. Čím rozsáhlejší je databáze, tím více odpovídají výsledky realitě. Spolehlivostní parametry se počítají právě z poskytnutých dat, výpočet se skládá ze dvou částí. První fáze představuje získávání hodnot atributů pro výpočet tzn. informace o počtu poruch a délce poruchy. Druhá fáze představuje samotný výpočet. V následujících odstavcích jsou popsány výpočty spolehlivostních parametrů [7].

Roční intenzita poruch udává statistickou spolehlivost prvku a odhadovanou pravděpodobnost za jakou dobu zařízení nebo prvek selže během celého roku používání. Jedná se o vztah mezi počtem porouchaných prvků a jejich celkovou dobou provozu než se u zařízení vyskytla porucha [7].

$$\lambda = \frac{N}{Z \cdot X}(year^{-1})$$

N - počet poruch
Z - pasportizační hodnota
X - konstanta pro měřené období (rok=1; 1/2 roku=0.5)

Střední doba trvání poruch je průměrná doba udávaná v hodinách, po které se měřený prvek navrátí do běžného provozního stavu. Čím menší je hodnota, tím rychleji dochází k obnovení očekávané činnosti prvku a tím více je měřený prvek spolehlivější. Nezbytným parametrem pro výpočet je počet poruch pro zvolené kritéria (napětí, typ zařízení, období, oblast). Tato kritéria jsou definována ve vstupních parametrech výpočtu. Čitatelem ve zlomku je celková doba trvání všech poruch [7].

$$\tau = \frac{\sum_{i=1}^{N} t_i}{N}(h)$$

t_i - délka poruchy
N - počet poruch

Výpis 1 obsahuje fragment metody, která se stará výpočet spolehlivostních parametrů u stávající aplikace. Počet poruch, pasportizační hodnota, délka poruch, roční intenzita poruch a střední doba trvání poruch jsou uloženy ve dvojrozměrné matici, kde řádky matice tvoří vybrané distribuční oblasti. Pro každý zvolený rok existuje právě jedna taková matice. K jednotlivým hodnotám této struktury lze přistupovat pomocí metod *GetValue* a *SetValue*. První parametr funkcí určuje matici pro daný rok. Druhý parametr určuje řádek a třetí parametr sloupec matice.

```
for (int i = 0; i < mReas.Count ; i++)
{
    double n = GetValue(year, i, ROW_N);
    double zxp = GetValue(year, i, ROW_ZxP);
    double time = GetValue(year, i, ROW_T);

    double lambda = 0.0;
    if (zxp > 0.0)
    {
        lambda = n / zxp;
    }
    SetValue(year, i, ROW_LAMBDA, lambda);

    double tau = 0.0;
    if (n > 0.0)
    {
        tau = time / n;
    }
    SetValue(year, i, ROW_TAU, tau);
}
```

Výpis 1: Výpočet spolehlivostních parametrů ve stávající aplikaci.

4 Reimplementace informačního systému

4.1 Specifikace požadavků

Následující podkapitola je zaměřena na specifikaci požadavků informačního systému. U níže zvýrazněných funkcí je potřeba reimplementovat datovou vrstvu, tak aby všechny funkce využívaly SQL dotazy. Dotazy nebudou z důvodu minimalizace času provádět kompletní výpočty spolehlivostních parametrů, ale budou načítat pouze záznamy s nezbytnými atributy, ze kterých bude proveden výpočet až na aplikační vrstvě. Interakce s informačním systém je umožněna 2 aktérům: uživateli a administrátorovi.

K přístupu do IS se bude muset uživatel nejprve přihlásit pomocí přihlašovacího jména a hesla, které může být změněno. Vytváření účtů spadá pod roli hlavního administrátora. Administrátor má možnost vybrat uživateli novou roli. Role uživatele se projeví v nabídce dostupných distribučních oblastí. Například pokud má uživatel roli REAS1, pak může provádět výpočty pouze pro poruchy ze své distribuční oblasti. Další možností administrátora je nastavit filtry při dotazování.

Pasportizace bude exportovat do Excelu vybraný pasportizační soubor poskytnutý distribuční společností. Každý řádek pasportizačního souboru se skládá z typu zařízení, identifikačního čísla prvku a napětí, roku, distribuční oblasti a pasportizační hodnoty. Ke každému řádku pasportizace bude dopočítán příslušný počet poruch, roční intenzita poruch, střední doba trvání poruch a celková délka trvání poruch pro atributy T3 a T4.

Cílem **prvkové spolehlivosti** bude spočítat počet poruch, životnost prvků, roční intenzitu poruch, délku trvání poruch, a střední dobu trvání poruch pro libovolně zvolené prvky, distributory, napětí a typ zařízení. Pro efektivní získávání pasportizační hodnoty bude vytvořena nová tabulka v databázi, do které budou naimportovány všechny záznamy z pasportizačních souborů. Z vypočítaných spolehlivostních parametrů bude vykreslen graf roční míry intenzity poruch a střední doby trvání poruch.

Výstupem funkce **spolehlivostních parametrů** bude roční intenzita poruch a střední doba trvání poruch vybraného prvku s daným napětím.

Provedením funkce **příčiny události** dojde ke spočítání počtu poruch daného typu zařízení pro všechny příčiny událostí seskupené podle roků a distributorů. Vypočtené hodnoty budou exportovány do Excelu a z výsledků bude vykreslen graf.

Četnost trvání poruchy bude mít za úkol spočítat počet poruch a vykreslit graf, s délkou trvání v minutách mezi určitými intervaly.

Dotazování slouží pro filtrování a zobrazování poruch přímo ve webové aplikaci. Původní verze informačního systému nabízí možnost exportovat záznamy do formátu XLS. Nová verze bude doplněná o export záznamů do CSV souboru.

4.2 Návrh aplikační vrstvy

Většina funkcí informačního systému bude provádět příkaz *SELECT*, tedy příkaz pro načítání záznamů. Takto získané záznamy je nutné převést na objekty. Pro tyto účely bude použit návrhový vzor data mapper. Data mapper specifikuje, že doménový objekt neobsahuje žádné CRUD operace a je tedy zcela nezávislý na databázi. O vytváření, editování a mazání doménových objektů z databáze se stará mapovací objekt. Výhodou tohoto vzoru je nezávislost doménového objektu na datovém modelu, a veškerá zodpovědnost je přesunuta na mapovací objekt [12]. SQL dotazy se budou vytvářet dynamicky za běhu aplikace. Hodnoty z formulářů vyplněné uživatelem se buď uloží do pomocných tabulek nebo se dosadí na místo proměnné v řetězci. Hodnoty uložené v pomocných tabulkách budou využity jako množina, která musí být obsažena ve výsledku. Toho lze docílit v SQL pomocí konstrukce IN.

4.3 Použité technologie při implementaci IS

Stávající informační systém je postaven na technologii .NET. Základem je programovací jazyk C# v kombinaci s C++. Informační systém je dostupný jako desktopová, tak i webová aplikace. Desktopová aplikace je vytvořená ve Windows Forms, která je navíc doplněná o možnost importovat nová data do databáze, zatímco webová aplikace využívá framework ASP.NET. Frontend u webové aplikace tvoří kombinace standardního značkovacího jazyka HTML s kaskádovými styly CSS. Při reimplementaci systému budou kromě dalších .NET technologií použity i knihovny třetích stran. Příkladem je knihovna pro práci s databázovým systémem SQLite. Další použitou knihovnou je CsvHelper [26], která nabízí programátorovi metody pro zápis a čtení CSV souborů. Pro práci na uživatelském rozhraní bude použita sada nástrojů Bootstrap [27], která obsahuje návrhářské šablony pro tvorbu webu.

4.4 Objektový přístup k datové vrstvě

Informační systém dokáže přistupovat k databázovému systému SQLite nebo SQL Serveru, proto je nutné zavést proxy třídu *OutageTableProxy* (viz. výpis 2), která si vybere patřičný DAO (data access abject). Proxy třída podle zvoleného nastavení v konfiguračním souboru, použije instanční třídu pro SQLite nebo SQL Server. Zároveň obsahuje abstraktní metody, které musí konkrétní třídy *OutageTableSQLite* a *OutageTableMSSQL* implementovat. V samotném programu se volají statické metody proxy třídy.

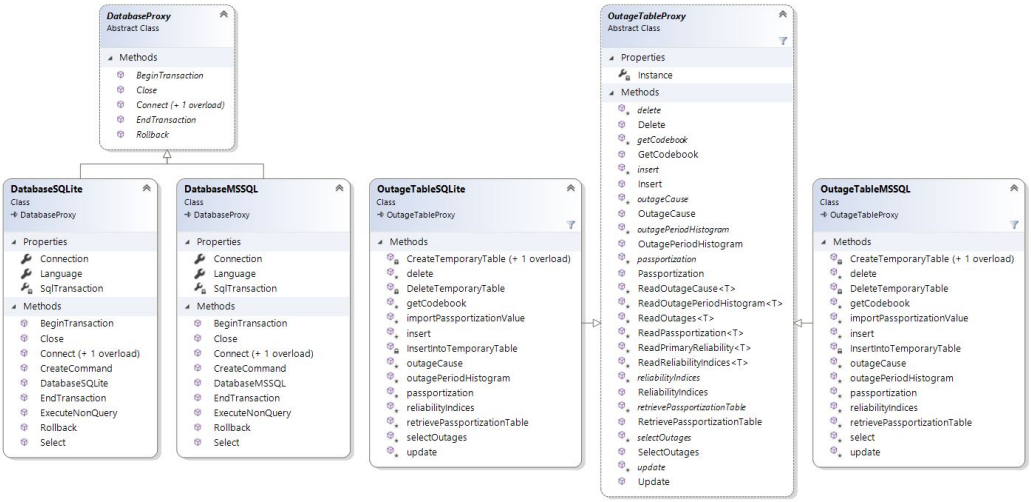
```
public abstract class OutageTableProxy
{
    // Instance obsahující referenci na konkrétní třídu pro MS SQL nebo SQLite.
    private static OutageTableProxy Instance
    {
        get
        {
            if (ConfigurationManager.AppSettings["DBMS"].ToLower()=="sqlite")
            {
                return new SQLite.OutageTableSQLite();
            }
            else
            {
                return new MSSQL.OutageTableMSSQL();
            }
        }
    }

    //Konkrétní třídy musí implementovat abstraktní metody
    protected abstract int insert(Outage outage);

    //Statické metody volají konkrétní metody pro SQLite nebo MS SQL.
    public static int Insert(Outage outage)
    {
        return Instance.insert(outage);
    }
}
```

Výpis 2: Proxy třída zastupující DAO pro SQLite a MS SQL.

Na obrázku 3 je uveden třídní diagram datové vrstvy. Abstraktní třída *DatabaseProxy* slouží jako bázová třída pro konkrétní třídy *DatabaseSQLite* a *DasatabaseMSSQL*. Tyto dvě třídy musí implementovat abstraktní metody z bázové třídy. Příkladem je metoda *Connect()* zajišťující spojení s databázovým systémem.



Obrázek 3: Třídní digram datové vrstvy.

4.5 Návrh datové vrstvy

Prvním úkolem je navrhnout relaci pro poruchy a naplnit ji záznamy ze stávající databáze. Zároveň je potřeba vytvořit relaci, do které budou naimportovány všechny záznamy z paspor-tizačních souborů. V našem případě použijeme databázové systémy MS SQL Server a SQLite. Oba DBMS jsou specifické a bude potřeba použít jejich podporované datové typy.

4.6 Datová analýza

Tabulka 5 zobrazuje schéma relace *Outage* a rozdíly v datových typech mezi SQL Serverem a SQLite. Tabulka 6 je schématem relace *Passportization*. U SQLite je vhodné upozornit na dynamičnost datových typů. Datový typ TEXT, který slouží primárně pro ukládání řetězců, umožňuje zpracovávat datum a čas ve formátu (YYYY-MM-DD HH:MM:SS.SSS). V takovémto formátu lze provádět klasické SQL operace jako <, >, =, BETWEEN. Velikost datového typu INTEGER je místo standardních 32 bitů dynamicky rozšiřitelná až na 8 B.

Pořadí	Název atributu	SQL Server datový typ	SQLite datový typ
1	distributor	int	INTEGER
2	event_order	bigint	INTEGER
3	event_type	int	INTEGER
4	distribution_point	int	INTEGER
5	area	int	INTEGER
6	network_type	int	INTEGER
7	network_voltage	int	INTEGER
8	equipment_voltage	int	INTEGER
9	original_event_order	int	INTEGER
10	event_cause	int	INTEGER
11	equipment_type	int	INTEGER
12	damaged_equipment	int	INTEGER
13	damaged_equipment_type	int	INTEGER
14	amount	int	INTEGER
15	short_type	int	INTEGER
16	producer	int	INTEGER
17	production_date	int	INTEGER
18	T0	datetime2	TEXT
19	T1	datetime2	TEXT
20	T2	datetime2	TEXT
21	T3	datetime2	TEXT
22	T4	datetime2	TEXT
23	TZ	datetime2	TEXT
24	P1	int	INTEGER
25	P2	int	INTEGER
26	D1	int	INTEGER
27	D2	int	INTEGER
28	Z1	int	INTEGER
29	Z2	int	INTEGER
30	LxT	int	INTEGER
31	failure_type	int	INTEGER

Tabulka 5: Relace *Outage*.

Název atributu	SQL Server datový typ	SQLite datový typ
d_p_s	char	TEXT
equipment	int	INTEGER
equipment_voltage	int	INTEGER
year	int	INTEGER
reas	int	INTEGER
value	float	REAL

Tabulka 6: Relace *Passportization*.

4.7 Funkční analýza

4.7.1 Seznam funkcí

- Export pasportizace
- Výpočet prvkové spolehlivosti
- Výpočet spolehlivostních parametrů
- Výpočet poruch seskupených podle příčiny událostí
- Výpočet poruch seskupených podle délky trvání
- Dotazování

4.7.2 Detailní popis funkcí

Cílem funkce pasportizace je exportovat vybraný pasportizační soubor do Excelu, který je navíc doplněný o spolehlivostní parametry. Třída ve výpisu 3 slouží jako doménový objekt pro export pasportizace. Vlastnosti D_P_S, EquipmentType, EquipmentVoltage, Year, Distributor a ZxP jsou získány pomocí metody 4.

```
public class PassportizationExport
    {
        public char D_P_S { get; set; }
        public Codebook EquipmentType { get; set; }
        public Codebook EquipmentVoltage { get; set; }
        public int Year { get; set; }
        public Codebook Distributor { get; set; }
        public string ZxP { get; set; }
        public int N { get; set; }
        public double Lambda => double.TryParse(ZxP, out double result) && result != 0 ? N / result : 0;
        public double T3_T0 { get; set; }
        public double TauT3_T0 => N == 0 ? 0 : T3_T0 / N;
        public double T4_T0 { get; set; }
        public double TauT4_T0 => N == 0 ? 0 : T4_T0 / N;
    }
```

Výpis 3: Doménový objekt pro export pasportizace.

Metoda ve výpisu 4 načte všechny záznamy z příslušného pasportizačního souboru. Všechny pasportizační soubory jsou uloženy na serveru v adresáři ~/projects/data/relnet/reas.

```
List<string> passportizationRows = new List<string>();
using (StreamReader streamReader = new StreamReader(fileName))
{
    while (streamReader.Peek() >= 0)
        passportizationRows.Add(streamReader.ReadLine());
}
```

Výpis 4: Algoritmus pro získání záznamů z pasportizačního souboru.

Následně je potřeba získat počet poruch a délku trvání poruch pro atributy T3 a T4 (N, T3_T0 T4_T0). Tyto 3 atributy jsou získány pomocí SQL dotazu ve výpisu 5.

```
select cb.id, c04.id,
(
   (
   select count(*)
   from Outage o
   where o.equipment_voltage = c04.id
   and o.event_type = 1
   and o.failure_type in (2)
   and o.distributor = 2
   and YEAR(o.T0) = 2002
   and o.damaged_equipment = cb.id
   )
) as [N],
(
   cast
   (
         (
            select sum(suma)
            from
            (
            select DATEDIFF(MINUTE, o.T0, o.T3) / 60.0 as suma from Outage o
            where o.equipment_voltage = c04.id
            and o.event_type = 1
            and o.failure_type in (2)
            and o.distributor = 2
            and YEAR(o.T0) = 2002
            and o.damaged_equipment = cb.id
            and o.T0 < o.T3
            and DATEDIFF(MINUTE, o.T0, o.T3) between 3 and 43200
            )
            T3
         )as float
   )
) as [T3-T0(h)],
(
   cast
   (
         (
            select sum(suma)
            from
            (
            select DATEDIFF(MINUTE, o.T0, o.T4) / 60.0 as suma from Outage o
            where o.equipment_voltage = c04.id
            and o.event_type = 1
            and o.failure_type in (2)
            and o.distributor = 2
            and YEAR(o.T0) = 2002
            and o.damaged_equipment = cb.id
            and o.T0 < o.T4
            and DATEDIFF(MINUTE, o.T0, o.T4) between 3 and 43200
            )
            T4
         )as float
      )
) as [T4-T0(h)]
from c_08 cb, c_04 c04
group by cb.id, c04.id
order by cb.id, c04.id
```

Výpis 5: SQL dotaz pro získání potřebných atributů dané pasportizace.

V dotazu byly zvoleny jako vstupní parametry poškozené zařízení, distribuční oblast 2 a rok 2002.

Jakmile instance třídy *PassportizationExport* inicializuje vlastnosti N, T3_T0, T4_T0, lze z těchto vlastností dopočítat spolehlivostní parametry (Lambda, TauT3_T0, TauT4_T0) viz. doménový objekt 3. Objekt je následně exportován do Excelu, tak jak je uvedeno ve výpisu 6.

```
mExcel.SetValue(excelRowIndex, 1, passportizationItem.D_P_S.ToString());
mExcel.SetValue(excelRowIndex, 2, (int)passportizationItem.EquipmentType.Id);
mExcel.SetValue(excelRowIndex, 3, (int)passportizationItem.EquipmentVoltage.Id);
mExcel.SetValue(excelRowIndex, 4, passportizationItem.Year);
mExcel.SetValue(excelRowIndex, 5, (int)passportizationItem.Distributor.Id);
mExcel.SetValue(excelRowIndex, 6, passportizationItem.ZxP);
mExcel.SetValue(excelRowIndex, 7, passportizationItem.N);
mExcel.SetValue(excelRowIndex, 8, passportizationItem.Lambda.ToString("F3"));
mExcel.SetValue(excelRowIndex, 9, passportizationItem.TauT4_T0.ToString("F3"));
mExcel.SetValue(excelRowIndex, 10, passportizationItem.T4_T0.ToString("F3"));
mExcel.SetValue(excelRowIndex, 11, passportizationItem.TauT3_T0.ToString("F3"));
mExcel.SetValue(excelRowIndex, 12, passportizationItem.T3_T0.ToString("F3"));
```

Výpis 6: Export pasportizace do Excelu.

Funkce prvkové spolehlivosti má za úkol spočítat spolehlivostní parametry pro vybrané prvky. Doménový objekt prvkové spolehlivosti je uveden ve výpisu 7. Vlatnosti Year, Reas, N, a T jsou získány z SQL dotazu 8. Vlastnost P, která slouží jako konstanta pro měřené období závisí na podmínce: Year != DateTime.Now.Year ? 1 : 0.5. Vlastnost ZxP je vyjádřena jako součin hodnot vlastností Z a P. Spolehlivostní parametry Lambda a Tau jsou vypočítány ze získaných hodnot vlastností. Takto vytvořené instance třídy jsou uloženy do původní struktury *PassportizationTableModel*, která se používá při práci s Excelem.

```
public class PrimaryReliability
{
    public int Year { get; set; }
    public int Reas { get; set; }
    public int N { get; set; }
    public double Z { get; set; }
    public double P => Year != DateTime.Now.Year ? 1 : 0.5;
    public double ZxP => Z * P;
    public double Lambda => ZxP == 0 ? 0 : N / ZxP;
    public double T { get; set; }
    public double Tau => N == 0 ? 0 : T / N;
}
```

Výpis 7: Doménový objekt prvkové spolehlivosti.

U výpočtu prvkové spolehlivosti může uživatel zvolit více prvků a distribučních oblastí. Jelikož by nebylo možné dosadit více hodnot za jednu proměnnou v dotazu, je potřeba využít pomocnou tabulku, ve které budou uloženy všechny vybrané hodnoty. Hodnoty uložené v pomocných tabulkách je možné zahrnout do dotazu pomocí konstrukce IN.

```
select YEAR(o.T0) as [year], c01.id as [reas],
(
   select count(*)
   from Outage o2
   where o2.equipment_voltage in (select equipment_voltage from Equipment_voltage_TemporaryTable)
   and o2.distributor = c01.id
   and YEAR(o2.T0) = YEAR(o.T0)
   and o2.event_type = 1
   and o2.failure_type in (2)
   and o2.damaged_equipment in (select equipment from Equipment_type_TemporaryTable)
) as [N],
(
   select sum(p.value)
   from Passportization p
   where p.equipment_voltage in (select equipment_voltage from Equipment_voltage_TemporaryTable)
   and p.reas = c01.id
   and p.year = YEAR(o.T0)
   and p.d_p_s = 'p'
   and p.equipment in (select equipment from Equipment_type_TemporaryTable)
) as [Z],
(
   cast
   (
      (
         select sum(suma)
         from
         (
         select DATEDIFF(MINUTE, o2.T0, o2.T3) / 60.0 as suma from Outage o2
         where o2.equipment_voltage in (select equipment_voltage from Equipment_voltage_TemporaryTable)
         and o2.distributor = c01.id
         and o2.event_type = 1
         and o2.failure_type in (2)
         and YEAR(o2.T0) = YEAR(o.T0) and o2.T3 is not null and o2.T0 < o2.T3
         and o2.damaged_equipment in (select equipment from Equipment_type_TemporaryTable)
         and DATEDIFF(MINUTE, o2.T0, o2.T3) between 3 and 43200
         union all
         select DATEDIFF(MINUTE, o2.T0, o2.T4)/ 60.0 as suma from Outage o2
         where o2.equipment_voltage in (select equipment_voltage from Equipment_voltage_TemporaryTable)
         and o2.distributor = c01.id
         and o2.event_type = 1
         and o2.failure_type in (2)
         and YEAR(o2.T0) = YEAR(o.T0) and o2.T3 is null and o2.T0 < o2.T4
         and o2.damaged_equipment in (select equipment from Equipment_type_TemporaryTable)
         and DATEDIFF(MINUTE, o2.T0, o2.T4) between 3 and 43200
         )
         T
      )as float
   )
) as [t(h)]
from Outage o, c_01 c01
where YEAR(o.T0) between 2000 and 2001
and c01.id in (select distributor from Distributor_TemporaryTable)
group by YEAR(o.T0), c01.id
order by YEAR(o.T0), c01.id
```

Výpis 8: SQL dotaz pro získání prvkové spolehlivosti.

Vstupní parametry dotazu: roky 2000 - 2001,
poškozené zařízení typu kabel (Equpment_type_TemporaryTable),
napětí zařízení 22kV (Equipment_voltage_TemporaryTable),
distribuční společnost 10 (Distributor_TemporaryTable).

Výpis 9 představuje doménový objekt spolehlivostních parametrů. Instance třídy ReliabilityIndices inicializuje vlastnosti EquipmentType, EquipmentVoltage, N, ZxP a T z SQL dotazu, který je uveden ve výpisu 10. Dotaz samotný výpočet spolehlivostních parametrů neprovádí, ale získává pouze nezbytné atributy, ze kterých jsou spolehlivostní parametry Lambda a Tau vypočítány (viz. výpis 9).

```
public class ReliabilityIndices
{
    public Codebook EquipmentType { get; set; }
    public Codebook EquipmentVoltage { get; set; }
    public int N { get; set; }
    public double ZxP { get; set; }
    public double Lambda => ZxP == 0 ? 0 : N / ZxP;
    public double T { get; set; }
    public double Tau => N == 0 ? 0 : T / N;
}
```

Výpis 9: Doménový objekt spolehlivostních parametrů.

Výpočet spolehlivostních parametrů využívá stejně jako předchozí funkce pomocné tabulky. Do pomocných tabulek se ukládají uživatelem vybrané prvky, distribuční oblasti a hodnoty napětí. Výpočet spolehlivostní parametrů se počítá pro všechny dostupné letopočty. Jedná se tedy o letopočty od roku 2000 až po současný rok.

```
select cb.id, cb.value, c04.id, c04.value,
(
   (
   select count(*)
   from Outage o
   where o.equipment_voltage = c04.id
   and o.event_type = 1
   and o.failure_type in (2)
   and o.distributor in (select distributor from Distributor_TemporaryTable)
   and YEAR(o.T0) between 2000 and 2019
   and o.damaged_equipment = cb.id
   )
) as [N],
(
   select sum(suma)
   from
   (
      select value as suma from Passportization p
      where p.equipment_voltage = c04.id
      and p.reas in (select distributor from Distributor_TemporaryTable)
      and p.year between 2000 and 2019 and p.year != YEAR(GETDATE())
      and p.equipment = cb.id
      and p.d_p_s = 'p'
      and 0 !=
      (
      select count(*)
      from Outage o
      where o.equipment_voltage = p.equipment_voltage
      and o.distributor = p.reas
      and YEAR(o.T0) = p.year
```

```
        and o.event_type = 1
        and o.failure_type in (2)
        and o.damaged_equipment = cb.id
        )
        union all
        select 0.5 * value as suma from Passportization p
        where p.equipment_voltage = c04.id
        and p.reas in (select distributor from Distributor_TemporaryTable)
        and p.year = YEAR(GETDATE())
        and p.equipment = cb.id
        and p.d_p_s = 'p'
        and 0 !=
        (
        select count(*)
        from Outage o
        where o.equipment_voltage = p.equipment_voltage
        and o.event_type = 1
        and o.failure_type in (2)
        and o.distributor = p.reas
        and YEAR(o.T0) = p.year
        and o.damaged_equipment = cb.id
        )
    )
    T
) as [ZxP],
(
    cast
    (
        (
            select sum(suma)
            from
            (
            select DATEDIFF(MINUTE, o.T0, o.T3) / 60.0 as suma from Outage o
            where o.equipment_voltage = c04.id
            and o.event_type = 1
            and o.failure_type in (2)
            and o.distributor in (select distributor from Distributor_TemporaryTable)
            and (YEAR(o.T0) between 2000 and 2019) and o.T3 is not null and o.T0 < o.T3
            and o.damaged_equipment = cb.id
            and DATEDIFF(MINUTE, o.T0, o.T3) between 3 and 43200
            union all
            select DATEDIFF(MINUTE, o.T0, o.T4)/ 60.0 as suma from Outage o
            where o.equipment_voltage = c04.id
            and o.event_type = 1
            and o.failure_type in (2)
            and o.distributor in (select distributor from Distributor_TemporaryTable)
            and (YEAR(o.T0) between 2000 and 2019) and o.T3 is null and o.T0 < o.T4
            and o.damaged_equipment = cb.id
            and DATEDIFF(MINUTE, o.T0, o.T4) between 3 and 43200
            )
            T
        ) as float
    )
) as [T]
from c_08 cb, c_04 c04
where cb.id in (select equipment from Equipment_type_TemporaryTable)
and c04.id in (select equipment_voltage from Equipment_voltage_TemporaryTable)
group by cb.id, cb.value, c04.id, c04.value
order by cb.id, cb.value, c04.id, c04.value
```

Výpis 10: SQL dotaz pro získání nezbytných atributů pro výpočet spolehlivostních parametrů.

Vstupní parametry dotazu:
zařízení typu stožár a vodič (Equipment_type_TemporaryTable),
napětí zařízení 22 a 110kV (Equipment_voltage_TemporaryTable),
distribuční společnost 10 (Distributor_TemporaryTable).

Následující funkce má za úkol spočítat počet poruch a seskupit je podle příčiny události. Třída EventCause uvedená ve výpisu 11 reprezentuje doménový objekt příčiny událostí. Všechny vlastnosti objektu jsou získány z SQL dotazu.

```
public class EventCause
{
    public Codebook OutageCause { get; set; }
    public int Year { get; set; }
    public Codebook Reas { get; set; }
    public int N { get; set; }
}
```

Výpis 11: Doménový objekt příčiny událostí.

SQL dotaz ve výpisu 12 seskupí počet poruch pro vybrané období a distribuční oblasti podle příčiny událostí.

```
select c06.id, years.years, c01.id,
  (
  select count(*)
  from Outage o
  where o.event_type = 1
  and o.failure_type in (2)
  and o.distributor = c01.id
  and o.event_cause = c06.id
  and YEAR(o.T0) = years.years
  ) as [N]
from c_06 c06, c_01 c01, Year_TemporaryTable years
where c01.id in (select distributor from Distributor_TemporaryTable)
group by c06.id, years.years, c01.id
order by cast(c06.id as varchar (10)), years.years, c01.id
```

Výpis 12: SQL dotaz pro získání počtu poruch seskupených podle příčiny událostí.

Vstupní parametry dotazu: rok 2002 (Year_TemporaryTable),
distribuční společnost 10 (Distributor_TemporaryTable).

Metoda ve výpisu 13 ukazuje, jak je ze získaných záznamů vytvořen objekt příčiny události. Objekt je následně přidán do kolekce, která se dále používá při práci s Excelem.

```
protected static List<EventCause> ReadOutageCause<T>(T reader) where T : DbDataReader
{
  List<EventCause> outageCauseList = new List<EventCause>();

  while (reader.Read())
   {
     int i = -1;
     EventCause outageCause = new EventCause();
     outageCause.OutageCause.Id = reader.GetInt32(++i);
     outageCause.Year = reader.GetInt32(++i);
     outageCause.Distributor.Id = reader.GetInt32(++i);
     outageCause.N = reader.GetInt32(++i);
     outageCauseList.Add(outageCause);
   }

  return outageCauseList;
}
```

Výpis 13: Metoda vytvářející objekty z vygenerovaných záznamů.

Funkce četnost trvání poruchy má za úkol spočítat počet poruch a seskupit je podle určitých intervalů. Počet záznamů lze v SQL spočítat pomocí funkce *COUNT()*. Interval dvou časových atributů je možné v SQL Serveru zjistit pomocí funkce DATEDIFF(). Interval je počítán mezi atributy T0 a T3 nebo mezi atributy T0 a T4, pokud atribut T3 nemá hodnotu. SQL dotaz pro získání četnosti trvání poruchy je uveden ve výpis 14

```
select
(
   select sum(suma)
   from
   (
      select count(*) as suma from Outage o
      where YEAR(o.T0) in (select years from Year_TemporaryTable)
      and o.distributor in (select distributor from Distributor\_TemporaryTable)
      and o.T0 < o.T3 and o.T3 is not null
      and DATEDIFF(MINUTE, o.T0, o.T3) between 0 and 1
      union all
      select count(*) as suma from Outage o
      where YEAR(o.T0) in (select years from Year_TemporaryTable)
      and o.distributor in (select distributor from Distributor\_TemporaryTable)
      and o.T0 < o.T4 and o.T3 is null
      and DATEDIFF(MINUTE, o.T0, o.T4) between 0 and 1
   )
   T
) as [0 - 1min],
(
   select sum(suma)
   from
   (
      select count(*) as suma from Outage o
      where YEAR(o.T0) in (select years from Year_TemporaryTable)
      and o.distributor in (select distributor from Distributor\_TemporaryTable)
      and o.T0 < o.T3 and o.T3 is not null
      and DATEDIFF(MINUTE, o.T0, o.T3) between 2 and 3
      union all
      select count(*) as suma from Outage o
      where YEAR(o.T0) in (select years from Year_TemporaryTable)
      and o.distributor in (select distributor from Distributor\_TemporaryTable)
      and o.T0 < o.T4 and o.T3 is null
      and DATEDIFF(MINUTE, o.T0, o.T4) between 2 and 3
   )
   T
) as [2 - 3min],
(
   select sum(suma)
   from
   (
      select count(*) as suma from Outage o
      where YEAR(o.T0) in (select years from Year_TemporaryTable)
      and o.distributor in (select distributor from Distributor\_TemporaryTable)
      and o.T0 < o.T3 and o.T3 is not null
      and DATEDIFF(MINUTE, o.T0, o.T3) between 4 and 10
      union all
      select count(*) as suma from Outage o
      where YEAR(o.T0) in (select years from Year_TemporaryTable)
      and o.distributor in (select distributor from Distributor\_TemporaryTable)
      and o.T0 < o.T4 and o.T3 is null
      and DATEDIFF(MINUTE, o.T0, o.T4) between 4 and 10
   )
   T
) as [4 - 10min]
```

Výpis 14: SQL dotaz pro získání četnosti trvání poruchy.

Metoda ve výpisu 15 vrací SQL dotaz, který se použije při dotazování nad poruchami. Dotaz je vytvořen na základě vstupních hodnot, které uživatel vybral ve formuláři.

```
public static string SelectOutagesCommand
(
List<string> reasList, string voltage, string eventCause,
string queryCode, string equipment, string year, string type,
string producer, EcSafety.Component.DbUtil.QueryInterval queryInterval, OutagePeriod outagePeriod
)
        {
            StringBuilder stringBuilder = new StringBuilder();
            stringBuilder.AppendLine("select * from Outage where 1=1");

            //Reas
            if (!reasList.Contains("*"))
            {
                StringBuilder stringBuilderReas = new StringBuilder();
                stringBuilderReas.Append("(");
                for (int i = 0; i < reasList.Count; i++)
                {
                    stringBuilderReas.Append(reasList[i]);
                    if (i != reasList.Count - 1)
                        stringBuilderReas.Append(",");
                }
                stringBuilderReas.Append(")");
                stringBuilder.AppendLine($"and distributor in {stringBuilderReas}");
            }

            //Voltage
            if (voltage != "*")
                stringBuilder.AppendLine($"and equipment_voltage = {voltage}");

            //Event cause and hierarchy
            if (eventCause != "*")
                stringBuilder.AppendLine($"and event_cause in (select id from {TableAndAttributeConstants.C_06} c06 where c06.id like
                    CONCAT({eventCause}, '%') and id >= {eventCause})");

            //Event type
            stringBuilder.AppendLine($"and event_type = 1");

            //Equipment type and equipments
            if (queryCode == "Poškozené zařízení" && equipment != "*")
                stringBuilder.AppendLine($"and damaged_equipment = {equipment}");
            else if (queryCode == "Druh zařízení" && equipment != "*")
                stringBuilder.AppendLine($"and equipment_type = {equipment}");

            //Year
            if (year != "*")
                stringBuilder.AppendLine($"and YEAR(T0) = {year}");

            //Damaged equipment type
            if (type != "*")
                stringBuilder.AppendLine($"and damaged_equipment_type = {type}");

            //Producer
            if (producer != "*")
                stringBuilder.AppendLine($"and producer = {producer}");

            //Event failure type
            if (queryInterval.GetLowValue(30) == 1)
                stringBuilder.AppendLine($"and failure_type in (1,2)");
            else if (queryInterval.GetLowValue(30) == 2)
                stringBuilder.AppendLine($"and event_type = 2");

            return stringBuilder.ToString();
        }
```

Výpis 15: Metoda vracející SQL pro dotazování nad poruchami.

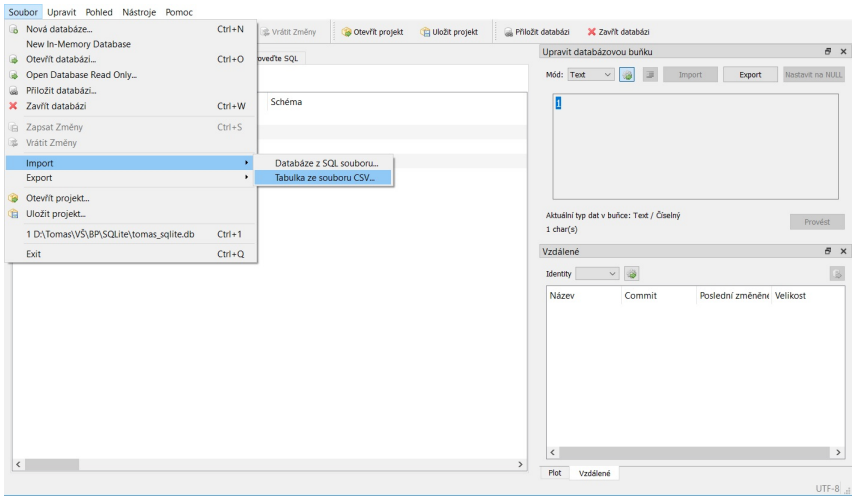
4.8 Import záznamů

Po vytvoření relace *Outage* je potřeba do ní naimportovat existující záznamy z dosavadního databázového systému. Všechny záznamy uložené v databázovém systému Radegast DB byly exportovány do souboru CSV, který je možno následně importovat do databázových systémů SQL Server a SQLite. SQL Server podporuje příkaz *bulk insert*, kterým lze tuto operaci provést.

```
bulk insert Outage
from 'outage.csv'
with
(
    firstrow =2,                //první importovaný řádek
    rowterminator='\n',         //oddělovač řádků
    fieldterminator =';',       //oddělovač sloupů
    CHECK_CONSTRAINTS   //validace cizích klíčů a podmínek
)
```

Výpis 16: Import záznamů z CSV souboru do databáze.

V případě SQLite lze záznamy importovat, pomocí několika kliknutí, přes DB Browser, nástroj sloužící pro správu SQLite databáze.



Obrázek 4: DB Browser, nástroj pro správu SQLite.

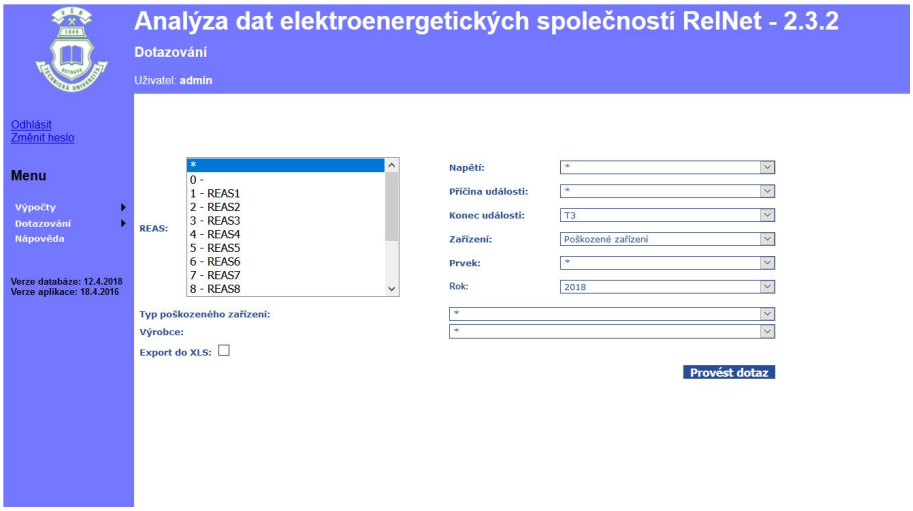
Tabulka 7 ukazuje prvních 5 záznamů v pasportizačním souboru p_reas02_2003. Všechny záznamy z pasportizačních souborů bylo nutno převést do relace *Passportization*.

d_p_s	kód	napětí	rok	reas	hodnota
d	1	1	2003	2	u
d	1	2	2003	2	0
d	1	3	2003	2	0
d	1	4	2003	2	0
d	1	5	2003	2	11137

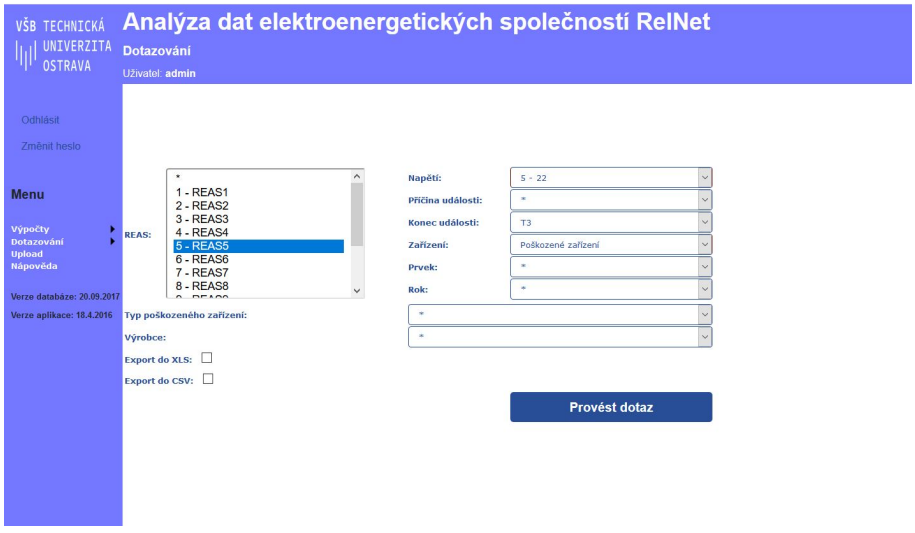
Tabulka 7: Pasportizační soubor p_reas02_2003.

4.9 Uživatelské rozhraní

Předposledním bodem bakalářské práce bylo přepsat uživatelské rozhraní, tak aby odpovídalo moderním trendům. Pro tento účel byl vybrán nástroj Bootstrap, který se v současnosti řadí mezi nejpopulárnější CSS frameworky. Bootstrap nabízí sadu hotových stylů, které lze aplikovat na HTML kód a vytvořit tak uživatelsky příjemné rozhraní. Bootstrap umožňuje úpravu typografie, formulářů, tlačítek, navigace a dalších komponent. V našem případě byly nahrazeny tlačítka a prvky formulářů. Mimo jiné bylo nahrazeno i zastaralé logo Vysoké školy báňské za logo současné, které bylo vytvořeno za účelem oslavy 170. výročí univerzity.



Obrázek 5: Uživatelské rozhraní původní aplikace.



Obrázek 6: Uživatelské rozhraní s aplikovaným frameworkem Bootstrap.

4.10 Srovnání použitých databázových systémů

V závěru práce bylo potřeba provést srovnání použitých databázových systémů a porovnat jejich efektivitu při dotazování. Podle naměřených časů lze určit zda reimplementace datové vrstvy byla přínosná i z pohledu doby strávené nad dotazováním. Pro každou operaci byly do dotazu vybrány volitelné atributy, které se běžně používají při výpočtech spolehlivostních parametrů. Tabulka 8 srovnává čas strávený vykonáváním jednotlivých operací pro použité databázové systémy.

	Čas vykonání operace v [ms]		
Funkce	Radegast DB	SQL Server	SQLite
Pasportizace	25[ms]	57[ms]	105757[ms]
Prvková spolehlivost	275[ms]	2643[ms]	22407[ms]
Spolehlivostní parametry	185[ms]	514[ms]	10167[ms]
Příčina událostí	102[ms]	830[ms]	9752[ms]
Četnost trvání poruch	97[ms]	238[ms]	8194[ms]
Dotazování	89[ms]	550[ms]	180[ms]

Tabulka 8: Srovnání použitých databázových systémů.

Měření jednoznačně prokázalo, že SQL Server disponuje efektivnějším plánovačem dotazů než SQLite i přes to, že v obou případech byly použity téměř totožné dotazy. V případě SQLite byla operace *prvková spolehlivost* byla vykonávána 8x déle a operace *spolehlivostní parametry* dokonce 20x déle než u SQL Serveru. Jako efektivnější se SQLite ukázal pouze u *dotazování*, u kterého SQL dotaz neobsahuje žádné poddotazy. Při relativním srovnání vykonaných dotazů s původním databázovým systém Radegast DB došlo ke zpomalení, které v případě SQL Serveru činí více něž 500%. Je nutno zmínit, že největší podíl na délce operace, od zpracování dotazu až po export, je v zápisu do souboru.

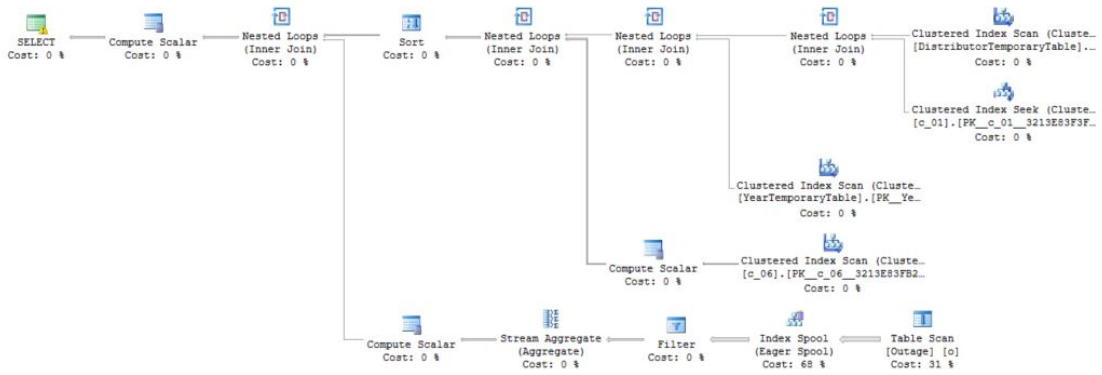
Výpis 17 je zaměřen na zásadní rozdíly v syntaxi mezi SQL serverem a SQLite. V zásadě se jedná o naprosto totožné dotazy, nicméně oba databázové systémy si implementují vlastní metody pro práci s datem a časem.

```
//SQL Server
select DATEDIFF(MINUTE, o.T0, o.T3) / 60.0 from Outage o
where o.distributor in (select distributor from Distributor\_TemporaryTable)
and YEAR(o.T0) = 2005

//SQLite
select cast((JulianDay(o.T3) - JulianDay(o.T0)) * 24 as real) from Outage o
where o.distributor in (select distributor from Distributor\_TemporaryTable)
and cast((strftime('%Y', o.T0)) as integer) = 2005
```

Výpis 17: Ukázka dotazu zaměřená na rozdíly v syntaxi mezi SQL Serverem a SQLite.

Obrázek 7 a tabulka 9 srovnávají plány vykonávání dotazu *příčina událostí*, který je uveden ve výpisu 12. Plán vykonávání u databázového systému SQLite umožňuje zobrazit pouze pořadí jednotlivých kroků daného dotazu. Plán vykonávání u SQL Serveru je možné zobrazit v grafické podobě, kde každý krok dotazu obsahuje detailní informace, jako je například časová náročnost. Z plánu vykonávání je patrné, že časově nejnáročnější je krok *index spool*, kdy dochází k formátování záznamů ze vstupu do dočasného indexu, který se používá při vyhledávání záznamů u vnořeného dotazu.



Obrázek 7: Plán vykonávání dotazu příčina událostí - SQL Server.

pořadí	operace
1	SEARCH TABLE c_01 AS c01 USING INTEGER PRIMARY KEY (rowid=?)
2	USING ROWID SEARCH ON TABLE Distributor_TemporaryTable FOR IN-OPERATOR
3	SCAN TABLE c_06 AS c06
4	SCAN TABLE Year_TemporaryTable AS years
5	CORRELATED SCALAR SUBQUERY 1
6	SCAN TABLE Outage AS o
7	USE TEMP B-TREE FOR ORDER BY

Tabulka 9: Plán vykonávání dotazu příčina událostí - SQLite.

4.11 Testování

Tato podkapitola slouží pro zhodnocení úprav systému a srovnání výsledků použitých databázových systémů s původním databázovým systémem Redegast DB.

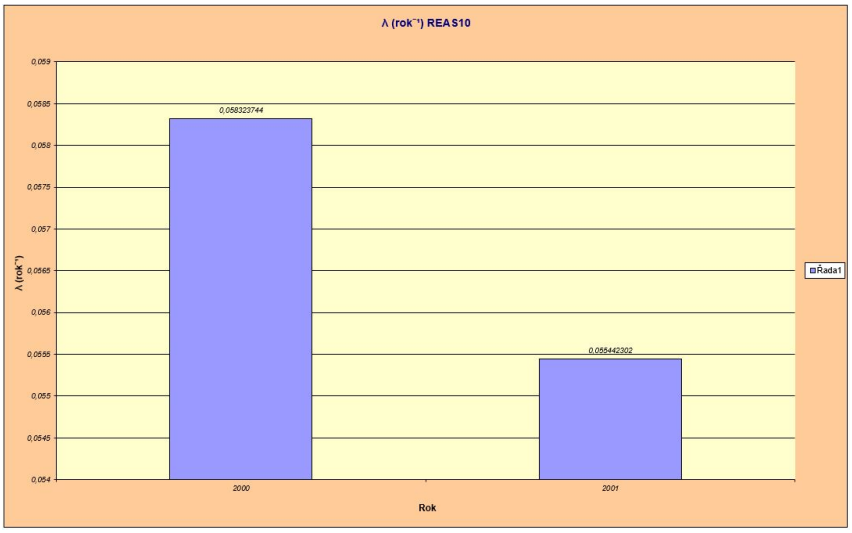
Tabulky 10 a 11 zobrazující výstup funkce prvkové spolehlivosti pro REAS10, poškozené zařízení typu kabel s napětím 22kV mezi letopočty 2001-2002. Výsledky se liší u celkové doby trvání poruch a střední doby trvání poruch, protože při reimplementaci byla objevena sémantická chyba, která způsobila, že do celkové délky všech poruch nebyla započítána poslední hodnota z vybrané množiny. Na obrázcích 8 a 9 jsou zobrazeny grafy roční míry poruchovosti a střední doby trvání poruch. Relativní srovnání střední doby trvání poruch v tomto případě činí rozdíl 0.39%.

	REAS10		
	2000	2001	2000-2001
N	28	37	65
Z	538	634,39	1172,39
P(rok)	1	1	1
ZxP	538	634,39	1172,39
λ (rok^{-1})	0,052	0,058	0,05
t (h)	364,90	635,26	1000,16
τ (h)	13,03	17,16	15.38

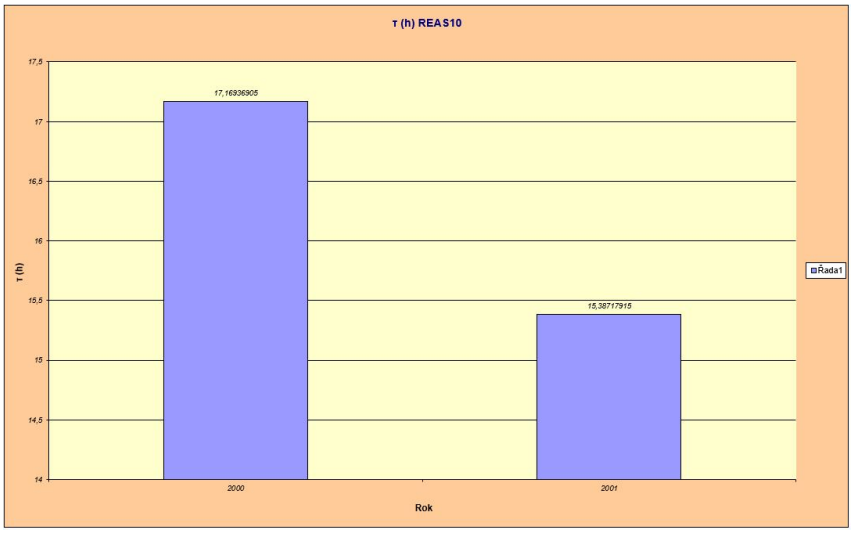
Tabulka 10: Prvková spolehlivost - SQL Server.

	REAS10		
	2000	2001	2000-2001
N	28	37	65
Z	538	634,39	1172,39
P(rok)	1	1	1
ZxP	538	634,39	1172,39
λ (rok^{-1})	0,052	0,058	0,05
t (h)	362,53	633,85	996,38
τ (h)	12,94	17,13	15,32

Tabulka 11: Prvková spolehlivost - Radegast DB.



Obrázek 8: Graf roční míry poruchovosti.



Obrázek 9: Graf střední doby trvání poruch.

Tabulky 12 a 13 jsou příkladem výstupu spolehlivostních parametrů pro distribuční oblast 10, poškozené zařízení stožár a vodič s napětím 22 a 110 kV. Výsledky se mírně liší, jelikož do systému nebyly importovány poslední záznamy z roku 2018 a 2019.

ID zařízení	Zařízení	ID napětí	Napětí(kV)	λ (rok^{-1})	τ (h)
1	stožár	5	22	0,001	9,869
1	stožár	7	110	0,002	24,792
2	vodič	5	22	0,022	5,249
2	vodič	7	110	0,002	6,653

Tabulka 12: Spolehlivostní parametry - SQL Server.

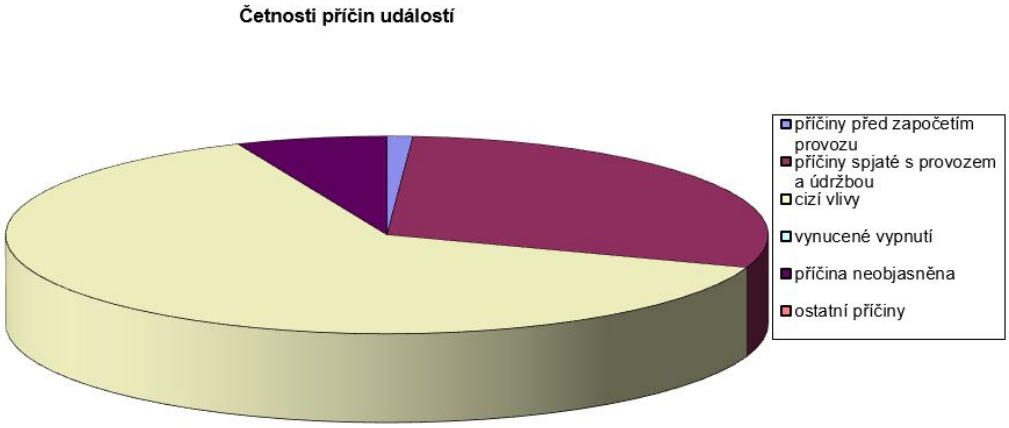
ID zařízení	Zařízení	ID napětí	Napětí(kV)	λ (rok^{-1})	τ (h)
1	stožár	5	22	0,001	13,314
1	stožár	7	110	0,003	22,317
2	vodič	5	22	0,017	7,429
2	vodič	7	110	0,001	9,917

Tabulka 13: Spolehlivostní parametry - Radegast DB.

V tabulce 14 je uveden počet poruch seskupený podle příčiny událostí pro poškozená zařízení z distribuční oblasti 10 a roku 2002. Součástí funkce je i vykreslení grafu příčiny událostí, který je uveden na obrázku 10.

Příčina události	Počet poruch
příčiny před započetím provozu	8
příčiny spjaté s provozem a údržbou	215
cizí vlivy	464
vynucené vypnutí	0
příčina neobjasněna	47
ostatní příčiny	0

Tabulka 14: Analýza příčin událostí.

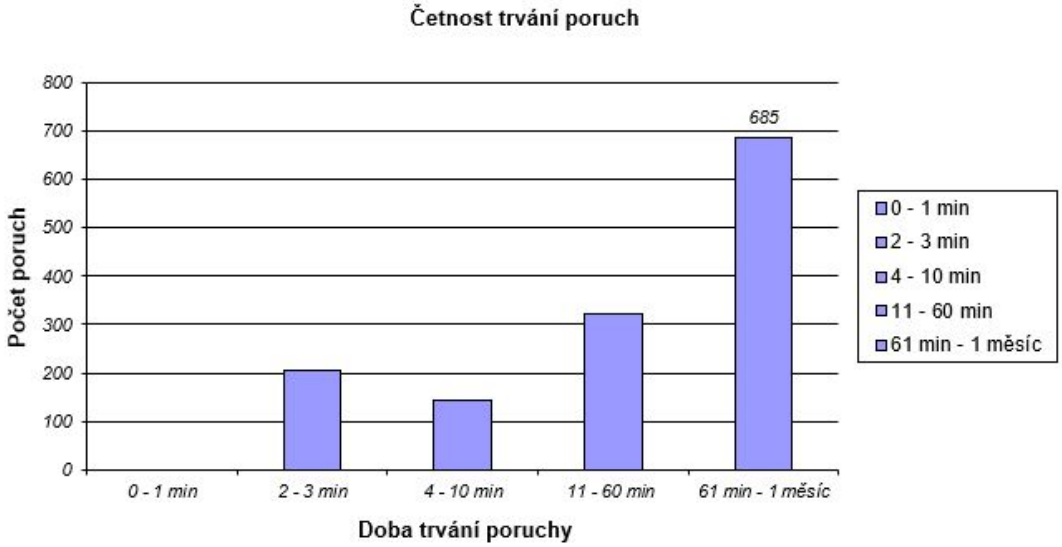


Obrázek 10: Graf příčin událostí

Rozdílné výsledky mezi původním a reimplementovaným systémem byly zaznamenány u operace *četnost trvání poruchy*. Rozdíl je částečně způsoben chybou v metodě stávajícího kódu, kde pro určovaní intervalu <x,y> [min] byl použit porovnávací operátor <. Pokud délka poruchy trvala například 1 minutu, pak byla porucha zařazena do intervalu <2,3>[min], jelikož algoritmus porovnává délku poruchy pouze s maximální hodnou intervalu. Rozdíl v celkovém počtu poruch je způsoben nově vloženými záznamy do stávající databáze. Ukázka výsledků je uvedena v tabulce 15. Z výsledků je vykreslen graf, který je znázorněn na obrázku 11.

	Počet poruch	
Délka poruchy	SQL Server	Radegast DB
0 - 1 [min]	7	0
2 - 3 [min]	9	205
4 - 10 [min]	15	143
11 - 60 [min]	211	321
61 [min] - 1 měsíc	492	685

Tabulka 15: Četnost trvání poruch pro rok 2002 a REAS10



Obrázek 11: Graf četnosti trvání poruch

Dotazování bylo doplněno o možnost exportovat záznamy do souboru CSV. Exportováním záznamů do textového souboru se několikanásobně zvětšila rychlost zápisu, která se projevila především při větším počtu záznamů. Textový soubor navíc není omezen maximálním počtem řádků. Tabulka 16 srovnává délku zápisu u XLS a CSV souborů pro různý počet exportovaných záznamů.

	Délka zápisu v [ms]	
Počet záznamů	XLS	CSV
79	519[ms]	2[ms]
734	1989[ms]	13[ms]
32454	110944[ms]	865[ms]

Tabulka 16: Délka zápisu do XLS a CSV

5 Závěr

Cílem bakalářské práce bylo nahradit stávající embedded databázový systém informačního systému za relační databázové systémy SQL Server a SQLite podporující jazyk SQL. Pro použité databázové systémy bylo provedeno srovnání, které jednoznačně prokázalo, že SQL Server je vybaven efektivnějším plánovačem dotazů než SQLite, a to i za předpokladu, že oba databázové systémy vyhodnocovaly téměř totožné dotazy. Problémem SQLite je vyhodnocování vnořených poddotazů, které zahrnují atributy z vnějších dotazů. Čas strávený nad vykonáváním takových dotazů je nepřijatelný a databázový systém SQLite je tak pro náš informační systém nepoužitelný. V relativním srovnání s původním databázovým systémem Radegast DB došlo u všech operací k rapidnímu zpomalení vykonávaných dotazů. Zpomalení některých dotazů činí až 500%. Nutno podotknout, že výhoda implementace za použití SQL dotazů není v efektivitě, ale spíše ve snadné rozšiřitelnosti systému o další výpočty.

V projektu byla aktualizována knihovna SQLite na nejnovější verzi, kterou využívá databáze uživatelů. Mimo jiné byla aktualizována i verze projektu na .NET Framework 4.7.2. Jediným problémem zůstává import nových záznamů do databáze. Původní embedded databáze řešila import přes desktopovou aplikaci. Jelikož build aplikace nebyl součástí zadání bakalářské práce, je její implementace stále v rozpracované fázi. Při reimplementaci systému byly objeveny sémantické chyby ve zdrojovém kódu, které jsou více rozvedeny v sekci testování.

Tato bakalářská práce vyžadovala znalost programovacího jazyka C#, dotazovacího jazyka SQL a celkovou dovednost při vývoji informačních systémů, včetně front-endového rozhraní. Zároveň jsem v teoretické části poukázal na nepoužívanější databázové systémy, jejich typy a architektury. Veškeré zdrojové kódy jsou součástí elektronické přílohy. Aplikaci je možné otestovat na školním webovém serveru, která je dostupná na adrese `http://srvfeia01/`.

Literatura

[1] JECHA, Tomáš. Lehký úvod – teorie databází: 1. díl - Lehký úvod – teorie databází. DotNETportal [online]. Česká republika: Tomáš Jecha, 2009, 12.12.2009 [cit. 2019-03-30]. Dostupné z: https://www.dotnetportal.cz/clanek/60/Lehky-uvod-teorie-databazi

[2] GARCIA-MOLINA, Hector, Jeffrey D. ULLMAN a Jennifer WIDOM. Database systems: the complete book. 2nd ed. Upper Saddle River, N.J.: Pearson Prentice Hall, c2009. ISBN 01-318-7325-3.

[3] Db-engines: Knowledge Base of Relational and NoSQL Database Management Systems [online]. USA: solid IT, 2019 [cit. 2019-03-30]. Dostupné z: https://db-engines.com/en/

[4] Architektura klient-server (Client–server model). ManagementMania [online]. Česká republika: ManagementMania, 2016, 4.11.2016 [cit. 2019-04-07]. Dostupné z: https://managementmania.com/cs/architektura-klient-server

[5] KOLÁŘ, Radim. Embedded databáze - Úvod. Root.cz [online]. Česká republika: Radim Kolář, 2004, 9. 3. 2004 [cit. 2019-03-30]. Dostupné z: https://www.root.cz/clanky/embedded-databaze-uvod/

[6] Codd, Edgar F. (June 1970). A Relational Model of Data for Large Shared Data Banks Communications of the ACM. 13 (6): 377–87. CiteSeerX 10.1.1.88.646. doi:10.1145/362384.362685

[7] GOŇO, Radomír, Michal KRÁTKÝ a Stanislav RUSEK. Analysis of Distribution Network Failure Databases. Przeglad elektrotechniczny. Warszawa: SIGMA-NOT, 2010,86(8), s. 168-171. ISSN 033-2097

[8] RDBMS dominate the database market, but NoSQL systems are catching up [online]. DB-Engines.com, 21 Nov 2013 [cit. 2019-04-07]

[9] KOCAN, Marek. Víte, co je SQL? Ne? Nevadí - dnes začínáme! Více na: https://www.zive.cz/clanky/vite-co-je-sql-ne-nevadi—dnes-zaciname/sc-3-a- 4320/default.aspx. Živě.cz [online]. Česká republika: Živě.cz, 1998, 26. října 1998 [cit. 2019-04-13]. Dostupné z: https://www.zive.cz/clanky/vite-co-je-sql-ne-nevadi—dnes- zaciname/sc-3-a-4320/default.aspx

[10] DELANEY, Kalen 2001. Inside Microsoft SQL Server 2000. Microsoft Press, a division of Microsoft Corporation, 2001

[11] SQL Server 2017 [online]. USA: Microsoft, 2017 [cit. 2019-04-07]. Dostupné z: https://www.microsoft.com/cs-cz/sql-server/sql-server-2017

[12] FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, c2003. ISBN 03-211-2742-0.

[13] ŽÁK, Karel. Historie relačních databází. Root.cz [online]. Česká republika: Root.cz, 2001, 19.10.2001 [cit. 2019-04-13]. Dostupné z: https://www.root.cz/clanky/historie-relacnich-databazi/

[14] SULLIVAN, Dan. NoSQL for Mere Mortals. 1. vyd. Upper Saddle River: Addison-Wessley, 2015. Kindle edition. ISBN 978-0-13-402231-2

[15] SQL Commands. Beginner SQL Tutorial [online]. USA: Beginner SQL Tutorial, 2007 [cit. 2019-04-14]. Dostupné z: https://beginner-sql-tutorial.com/sql-commands.htm

[16] Oracle Database [online]. USA: Oracle, 2018 [cit. 2019-04-14]. Dostupné z: https://www.oracle.com/cz/database/

[17] Oracle Database (Oracle DB): USA [online]. USA: Techopedia, 2011 [cit. 2019-04-16]. Dostupné z: https://www.techopedia.com/definition/8711/oracle-database

[18] PostgreSQL: The World's Most Advanced Open Source Relational Database [online]. USA: PostgreSQL Global Development Group, 2019 [cit. 2019-04-15]. Dostupné z: https://www.postgresql.org/

[19] PostgreSQL: Documentation: 10: 1. What is PostgreSQL? [online]. The PostgreSQL Global Development Group [cit. 2019-04-15]

[20] MySQL [online]. USA: MySQL, 2019 [cit. 2019-04-14]. Dostupné z: https://www.mysql.com/

[21] What is MySQL?: USA [online]. USA: Oracle, 2019 [cit. 2019-04-16]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.htmle

[22] SQLite [online]. USA: SQLite, 2019 [cit. 2019-04-14]. Dostupné z: https://www.sqlite.org/index.html

[23] SQLite - ultra lehké sql. Root.cz [online]. Česká republika: Root.cz, 2003, 4. 8. 2003 [cit. 2019-04-14]. Dostupné z: https://www.root.cz/clanky/sqlite-ultra-lehke-sql/

[24] MongoDB [online]. USA: MongoDB, 2019 [cit. 2019-04-14]. Dostupné z: https://www.mongodb.com/

[25] What is MongoDB? [online]. USA: Mongo DB, 2019 [cit. 2019-04-15]. Dostupné z: https://www.mongodb.com/what-is-mongodb

[26] CsvHelper: USA [online]. USA: CsvHelper, 2019 [cit. 2019-04-16]. Dostupné z: https://joshclose.github.io/CsvHelper/

[27] Bootstrap: USA [online]. USA: Bootstrap, 2019 [cit. 2019-04-16]. Dostupné z: https://getbootstrap.com/

[28] ANSI X3.135-1986. SQL86. 1986

[29] ISO/IEC 9075:1992. SQL-92. 1992

[30] Energetický regulační úřad [online]. Česká republika: Energetický regulační úřad, 2014 [cit. 2019-04-27]. Dostupné z: https://www.eru.cz/cs/

Seznam elektronických příloh

1. Číselníky
2. Zdrojové kódy aplikace